Notes on Iterative Closest Point (Point-to-Point, Point-to-Plane, GICP)

Gyubeom Edward Im*

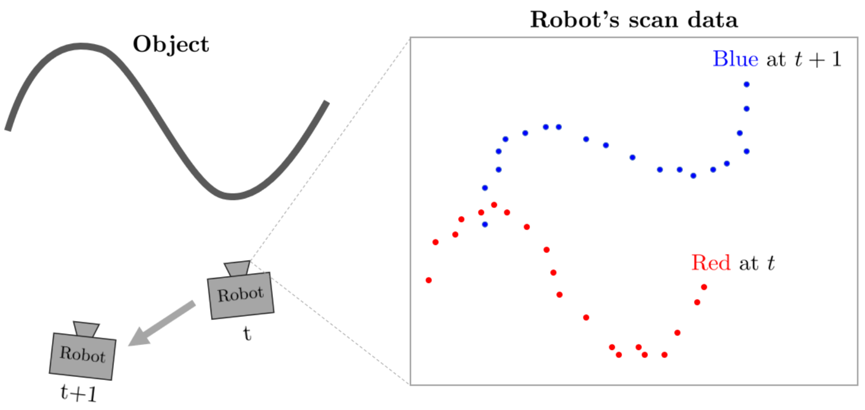
Contents

1 Introduction

Iterative Closest Point (ICP) is a method that, given two point-cloud sets, **searches for the nearest points** for each point and then **iteratively performs registration** based on them. ICP is mainly used to align 3D scan data in LiDAR SLAM, and variants such as Point-to-Point and Point-to-Plane exist.

*blog: alida.tistory.com, email: criterion.im@gmail.com

2 Example pointcloud data (2D)



First, consider a 2D point-cloud example. Assume a robot acquires the following data at times t and $t+1$ using a 2D scanner.

The 2D point-cloud coordinates of the points at time t (red) and at time $t+1$ (blue) are as follows.

Tip

sourcePoints = [[-19, -15], [-18, -10], [-15, -9], [-14, -7], [-11, -6], [-9, -5], [-7, -6], [-4, -8], [-1, -11], [0, -14], [1, -17], [5, -20], [9, -24], [10, -25], [13, -24], [14, -25], [17, -25], [19, -22], [22, -18], [23, -16]]

–

targetPoints = [[-12, -8], [-12, -2], [-10, 1], [-10, 4], [-9, 6], [-6, 7], [-3, 8], [-1, 8], [3, 6], [6, 5], [10, 3], [14, 1], [17, 1], [19, 0], [22, 1], [24, 2], [27, 4], [26, 7], [27, 11], [27, 15]]

3 Example pointcloud data (3D)

However, in practice we are typically given 3D point-cloud data. Assume a robot acquires the following data at times t and $t+1$ using a 3D scanner.

The 3D point-cloud coordinates of the points at time t (red) and at time $t+1$ (blue) are as follows.

Tip

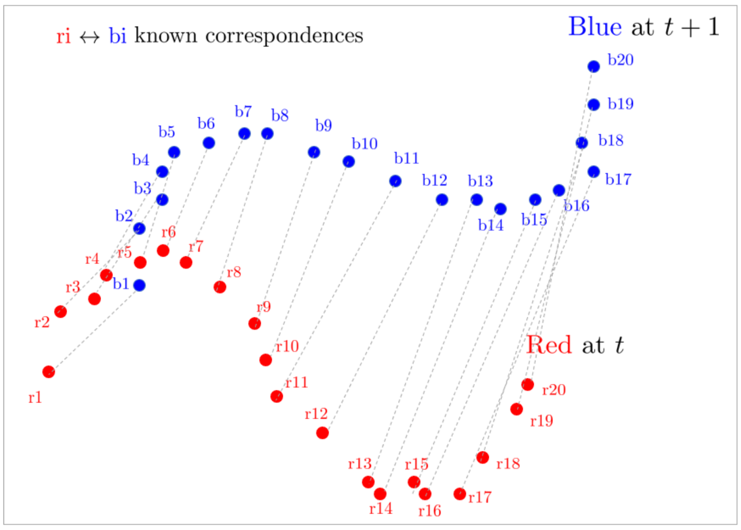
sourcePoints = [[-19, -15, 7], [-18, -10, 6], [-15, -9, 5], [-14, -7, 4], [-11, -6, 8], [-9, -5, 5], [-7, -6, 7], [-4, -8, 6], [-1, -11, 4], [0, -14, 6], [1, -17, 8], [5, -20, 7], [9, -24, 5], [10, -25, 6], [13, -24, 8], [14, -25, 5], [17, -25, 7], [19, -22, 6], [22, -18, 8], [23, -16, 7]]

–

targetPoints = [[-12, -8, 9], [-12, -2, 11], [-10, 1, 10], [-10, 4, 12], [-9, 6, 9], [-6, 7, 10], [-3, 8, 8], [-1, 8, 12], [3, 6, 11], [6, 5, 9], [10, 3, 8], [14, 1, 12], [17, 1, 11], [19, 0, 10], [22, 1, 8], [24, 2, 9], [27, 4, 11], [26, 7, 12], [27, 11, 9], [27, 15, 10]]

4 Point-to-point ICP

4.1 With known data associations



Before diving into ICP, consider the easiest case. If the association/correspondence $r_i \leftrightarrow b_i$ between the two point clouds is given in advance as in the figure, the registration problem has a closed-form solution and can be solved easily. (No initial guess, no iterative)

If we denote the red point cloud as $\mathbf{p}_t = [\mathbf{p}_{t,1}, \mathbf{p}_{t,2}, \cdots, \mathbf{p}_{t,n}]^\intercal$ and the blue point cloud as $\mathbf{p}_{t+1} = [\mathbf{p}_{t+1,1}, \mathbf{p}_{t+1,2}, \cdots, \mathbf{p}_{t+1,n}]$ the relation between the two point clouds can be written as follows.

$$\mathbf{p}_{t+1} \approx \mathbf{R}\mathbf{p}_t + \mathbf{t} \tag{1}$$

- The above equation holds for both 2D and 3D data $\mathbf{p}_t$. Therefore it holds for both $\mathbf{R} \in SO(2)$ and $\mathbf{R} \in SO(3)$.

The above equation can be converted into the following least-squares problem.

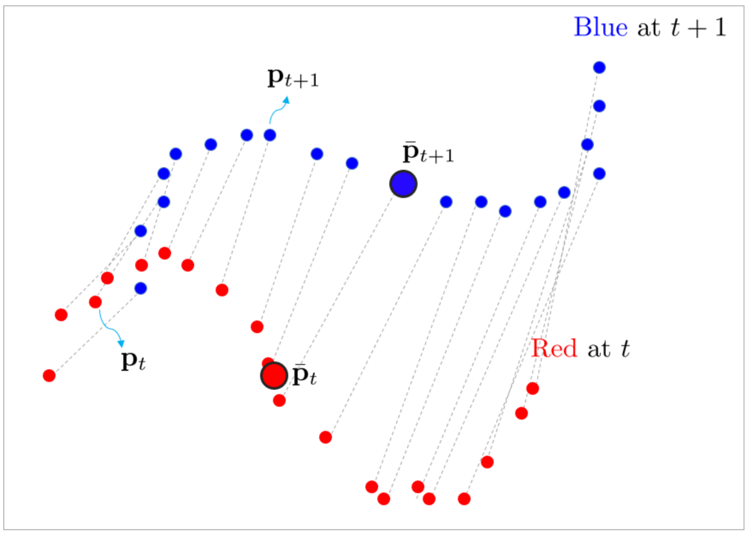
$$\boxed{\arg\min_{\mathbf{R},\mathbf{t}} \|\mathbf{p}_{t+1} - \mathbf{R}\mathbf{p}_t - \mathbf{t}\|^2} \tag{2}$$

This least-squares problem cannot be solved in the normal-equation form because it contains the nonlinear term $\mathbf{R}$. To address this, there are approaches such as solving via SVD of the covariance matrix or via nonlinear least squares (Gauss–Newton). We first explain the covariance-matrix SVD approach.

4.1.1 Covariance SVD-based solution

In short, after computing the covariance matrix $\mathbf{C}$ of the two point clouds, we can obtain the optimal rotation matrix $\mathbf{R}^*$ by performing SVD on it.

First, compute the centroids of the two point clouds.



$$\begin{aligned} \bar{\mathbf{p}}_t &= \frac{1}{n}\sum \mathbf{p}_{t,i} \\ \bar{\mathbf{p}}_{t+1} &= \frac{1}{n}\sum \mathbf{p}_{t+1,i} \end{aligned} \tag{3}$$

Next, for each point $\mathbf{p}$, compute $\mathbf{p}'$ by subtracting the centroid $\bar{\mathbf{p}}$.

$$\begin{aligned} \mathbf{p}'_t &= \mathbf{p}_t - \bar{\mathbf{p}}_t \\ \mathbf{p}'_{t+1} &= \mathbf{p}_{t+1} - \bar{\mathbf{p}}_{t+1} \end{aligned} \tag{4}$$

Using the above result, we can rewrite (2) as follows.

$$\arg\min_{\mathbf{R},\mathbf{t}} \|(\mathbf{p}'_{t+1} + \bar{\mathbf{p}}_{t+1}) - \mathbf{R}(\mathbf{p}'_t + \bar{\mathbf{p}}_t) - \mathbf{t}\|^2 \tag{5}$$

- $\mathbf{p}_{t+1} = \mathbf{p}'_{t+1} + \bar{\mathbf{p}}_{t+1}$
- $\mathbf{p}_t = \mathbf{p}'_t + \bar{\mathbf{p}}_t$

Here, the translation vector $\mathbf{t}$ between the two point clouds is set to the difference between the centroid at time $t+1$, $\bar{\mathbf{p}}_{t+1}$, and the centroid of the rotated point cloud at time t, $\mathbf{R}\bar{\mathbf{p}}_t$. In other words, **if the relative rotation between the two point clouds is corrected exactly, then the two point clouds are assumed to be separated by exactly t.**

$$\boxed{\mathbf{t} = \bar{\mathbf{p}}_{t+1} - \mathbf{R}\bar{\mathbf{p}}_t} \tag{6}$$

Substituting the above into (5) eliminates the $\mathbf{t}$ term and simplifies the problem to finding $\mathbf{R}$.

$$\arg\min_{\mathbf{R}} \|\mathbf{p}'_{t+1} - \mathbf{R}\mathbf{p}'_t\|^2 \tag{7}$$

Expanding the above yields the following. During expansion, terms cancel out and only the middle term remains related to $\mathbf{R}$.

$$\begin{aligned} \arg\min_{\mathbf{R}} \left\|\mathbf{p}'_{t+1} - \mathbf{R}\mathbf{p}'_t\right\|^2 &= \arg\min_{\mathbf{R}} \left[\mathbf{p}'^{\intercal}_{t+1}\mathbf{p}'_{t+1} - 2\mathbf{p}'^{\intercal}_{t+1}\mathbf{R}\mathbf{p}'_t + \mathbf{p}'^{\intercal}_t \underbrace{\mathbf{R}^\intercal\mathbf{R}}_{\mathbf{I}} \mathbf{p}'_t\right] \\ &= \arg\min_{\mathbf{R}} \left[-2\mathbf{p}'^{\intercal}_{t+1}\mathbf{R}\mathbf{p}'_t + C\right] \end{aligned} \tag{8}$$

Therefore, the optimization can be rewritten as follows. As the minus sign $(-)$ disappears, the argmin problem is converted into an argmax problem.

$$\begin{aligned}\mathbf{R}^* &= \arg\min_{\mathbf{R}} \Big[-2\mathbf{p}'^{\intercal}_{t+1}\mathbf{R}\mathbf{p}'_t \Big] \\ &= \arg\max_{\mathbf{R}} \Big[\mathbf{p}'^{\intercal}_{t+1}\mathbf{R}\mathbf{p}'_t \Big] \\ &= \arg\max_{\mathbf{R}} \Big[\mathrm{tr}(\mathbf{p}'^{\intercal}_{t+1}\mathbf{R}\mathbf{p}'_t) \Big] \\ &= \arg\max_{\mathbf{R}} \Big[\mathrm{tr}(\mathbf{R}\mathbf{p}'_t\mathbf{p}'^{\intercal}_{t+1}) \Big] \quad \leftarrow \because \mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA}) \\ &= \arg\max_{\mathbf{R}} \Big[\mathrm{tr}(\mathbf{RC}) \Big]\end{aligned} \tag{9}$$

- $\mathbf{C} = \mathbf{p}'_t\mathbf{p}'^{\intercal}_{t+1}$

In the third line above, we can see that $\mathbf{p}'^{\intercal}_{t+1}\mathbf{R}\mathbf{p}'_t = \mathbb{R}^{1\times n}\cdot\mathbb{R}^{n\times n}\cdot\mathbb{R}^{n\times 1} = \mathbb{R}$ is a scalar. **Since the final result is a scalar (= a 1×1 matrix), we can exploit useful properties of the trace, i.e., the sum of diagonal elements.** Using $\mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA})$, we swap the order by setting $\mathbf{A} = \mathbf{p}'^{\intercal}_{t+1}$ and $\mathbf{B} = \mathbf{R}\mathbf{p}'_t$.

In the fourth line above, $\mathbf{p}'_t\mathbf{p}'^{\intercal}_{t+1}$ is **exactly the definition of the covariance matrix of the two point clouds.** Thus we replace it with $\mathbf{C}$.

Tip

Covariance matrix of $\mathbf{x}, \mathbf{y}$

For two data sets $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]^{\intercal}$ and $\mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n]^{\intercal}$, the covariance matrix between the two data sets can be computed as follows.

1. Compute the mean of each data set

$$\begin{aligned}\bar{\mathbf{x}} &= \frac{1}{n}\sum \mathbf{x}_i \\ \bar{\mathbf{y}} &= \frac{1}{n}\sum \mathbf{y}_i\end{aligned} \tag{10}$$

2. Subtract the mean from each original data point

$$\begin{aligned}\mathbf{x}' &= \mathbf{x} - \bar{\mathbf{x}} \\ \mathbf{y}' &= \mathbf{y} - \bar{\mathbf{y}}\end{aligned} \tag{11}$$

3. Then the covariance matrix $\mathbf{C}_{\mathbf{xy}}$ is obtained as follows

$$\begin{aligned}\mathbf{C}_{\mathbf{xy}} &= \mathbf{x}'\mathbf{y}'^{\intercal} \\ &= (\mathbf{x} - \bar{\mathbf{x}})(\mathbf{y} - \bar{\mathbf{y}})^{\intercal}\end{aligned} \tag{12}$$

Therefore, to obtain the optimal $\mathbf{R}^*$, we need to solve the following.

$$\begin{aligned}\mathbf{R}^* &= \arg\max_{\mathbf{R}} \Big[\mathrm{tr}(\mathbf{RC}) \Big] \\ &= \arg\max_{\mathbf{R}} \Big[\mathrm{tr}(\mathbf{RUDV}^{\intercal}) \Big]\end{aligned} \tag{13}$$

The covariance matrix C cannot have a negative value in this context, so it is always a positive (semi-)definite matrix. Therefore, when we take the SVD of $\mathbf{C}$ in the second line above, all singular values are always greater than or equal to zero.

Tip

Lemma

For an arbitrary positive definite matrix $\mathbf{AA}^\intercal$ and an orthonormal matrix $\mathbf{B}$, the following Cauchy–Schwarz inequality holds.

$$\text{tr}(\mathbf{AA}^\intercal) \geq \text{tr}(\mathbf{BAA}^\intercal) \tag{14}$$

Proof

The Cauchy–Schwarz inequality for arbitrary vectors is as follows.

$$|\langle \mathbf{u}, \mathbf{v}\rangle \leq \|\mathbf{u}\| \cdot \|\mathbf{v}\| \tag{15}$$

- $\mathbf{u}, \mathbf{v}$: arbitrary vectors
- $\langle\cdot,\cdot\rangle$: vector inner product
- $\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a}\rangle}$: vector norm

$\text{tr}(\mathbf{BAA}^\intercal)$ can be expressed in vector form as follows

$$\begin{aligned} \text{tr}(\underline{\mathbf{BA}} \cdot \underline{\underline{\mathbf{A}^\intercal}}) &= \text{tr}(\underline{\underline{\mathbf{A}^\intercal}} \cdot \underline{\mathbf{BA}}) \\ &= \sum \mathbf{a}_i^\intercal \mathbf{B} \mathbf{a}_i \end{aligned} \tag{16}$$

Substituting the above into the Cauchy–Schwarz inequality yields the following.

$$\begin{aligned} \mathbf{a}_i^\intercal \mathbf{B} \mathbf{a}_i &\leq \sqrt{(\mathbf{a}_i^\intercal \mathbf{a}_i)(\mathbf{a}_i^\intercal \underbrace{\mathbf{B}^\intercal \mathbf{B}}_{\mathbf{I}} \mathbf{a}_i)} \\ &= \mathbf{a}_i^\intercal \mathbf{a}_i \end{aligned} \tag{17}$$

$$\mathbf{a}_i^\intercal \mathbf{B} \mathbf{a}_i \leq \mathbf{a}_i^\intercal \mathbf{a}_i \tag{18}$$

Therefore, we obtain the following lemma.

$$\boxed{\therefore \text{tr}(\mathbf{AA}^\intercal) \geq \text{tr}(\mathbf{BAA}^\intercal)} \tag{19}$$

4.1.2 Method 1 for $\mathbf{R}^*$

If we define $\mathbf{R} = \mathbf{VU}^\intercal$, then in (13) the orthonormal matrices $\mathbf{U}^\intercal\mathbf{U}$ cancel out, leaving only $\mathbf{VDV}^\intercal$. This allows us to form the $\mathbf{AA}^\intercal$ structure.

$$\boxed{\begin{aligned} \text{tr}(\mathbf{RUDV}^\intercal) &= \text{tr}(\mathbf{V} \underbrace{\mathbf{U}^\intercal\mathbf{U}}_{\mathbf{I}} \mathbf{DV}^\intercal) \\ &= \text{tr}(\mathbf{VDV}^\intercal) \\ &= \text{tr}(\mathbf{VD}^{\frac{1}{2}}\mathbf{D}^{\frac{1}{2}}\mathbf{V}^\intercal) \\ &= \text{tr}(\mathbf{VD}^{\frac{1}{2}})(\mathbf{D}^{\frac{1}{2}}\mathbf{V}^\intercal) \quad \leftarrow \mathbf{AA}^\intercal \text{ form} \\ &= \text{tr}(\mathbf{AA}^\intercal) \\ &\geq \text{tr}(\mathbf{R}'\mathbf{AA}^\intercal) \end{aligned}} \tag{20}$$

- $\mathbf{R}'$: an arbitrary orthonormal matrix
- $\mathbf{A} = \mathbf{VD}^{\frac{1}{2}}$

Rearranging gives the following.

$$\boxed{\text{tr}(\mathbf{RC}) = \text{tr}(\mathbf{VU}^\intercal\mathbf{C}) \geq \text{tr}(\mathbf{R}'\mathbf{AA}^\intercal) = \text{tr}(\mathbf{R}'\mathbf{RC})} \tag{21}$$

This means that when $\mathbf{R} = \mathbf{VU}^\intercal$, it yields a larger value than any other rotation matrix of the form $\mathbf{R}'\mathbf{R}$. Hence it is the argmax solution.

$$\boxed{\begin{aligned}&\mathbf{R} = \mathbf{V}\mathbf{U}^\intercal \\ &\mathbf{t} = \mathbf{R}\bar{\mathbf{p}}_t - \bar{\mathbf{p}}_{t+1}\end{aligned}} \tag{22}$$

4.1.3 Method 2 for $\mathbf{R}^*$

Looking again at (13), we must maximize the value inside the trace. As noted above, the covariance matrix $\mathbf{C}$ is positive (semi-)definite, so all singular values are nonnegative and thus all diagonal entries of $\mathbf{D}$ are nonnegative. Since $\mathbf{R}, \mathbf{U}, \mathbf{V}$ are orthonormal, multiplying them with the diagonal matrix always makes the trace value no greater than that of $\mathbf{D}$ itself.

$$\text{tr}(\mathbf{D}) \geq \text{tr}(\mathbf{R}\mathbf{U}\mathbf{D}\mathbf{V}^\intercal) \tag{23}$$

Using the trace property to swap the order, we get the following.

$$\begin{aligned}\mathbf{R}^* &= \arg\max_{\mathbf{R}} \left[\text{tr}(\underline{\underline{\mathbf{R}\mathbf{U}}} \cdot \underline{\mathbf{D}\mathbf{V}^\intercal})\right] \\ &= \arg\max_{\mathbf{R}} \left[\text{tr}(\underline{\mathbf{D}\mathbf{V}^\intercal} \cdot \underline{\underline{\mathbf{R}\mathbf{U}}})\right]\end{aligned} \tag{24}$$

If $\mathbf{R} = \mathbf{V}\mathbf{U}^\intercal$, the orthonormal matrices cancel out and the trace becomes maximal.

$$\begin{aligned}\mathbf{R}^* &= \arg\max_{\mathbf{R}} \left[\text{tr}(\mathbf{D}\mathbf{V}^\intercal\mathbf{R}\mathbf{U})\right] \\ &= \arg\max_{\mathbf{R}} \left[\text{tr}(\mathbf{D}\underbrace{\mathbf{V}^\intercal\mathbf{R}\mathbf{U}}_{\mathbf{I}})\right]\end{aligned} \tag{25}$$

- $\mathbf{R} = \mathbf{V}\mathbf{U}^\intercal$

Therefore, we recover the same $\mathbf{R}$ as in (22) derived in method 1.

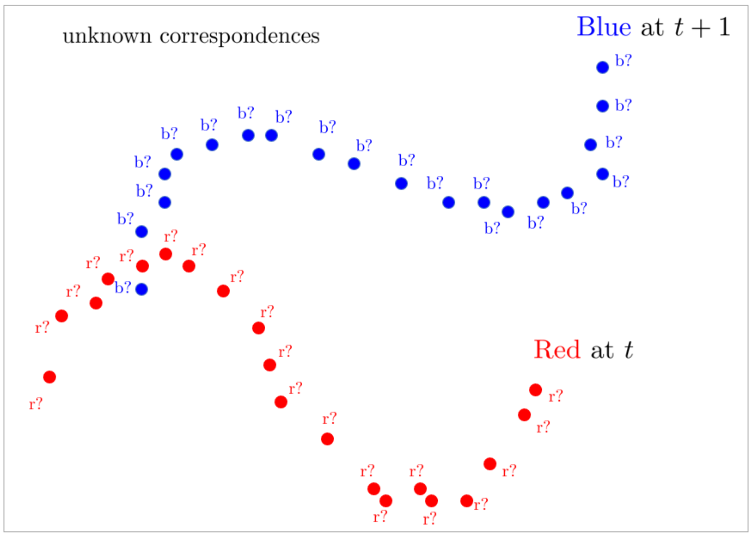
$$\boxed{\begin{aligned}&\mathbf{R} = \mathbf{V}\mathbf{U}^\intercal \\ &\mathbf{t} = \bar{\mathbf{p}}_{t+1} - \mathbf{R}\bar{\mathbf{p}}_t\end{aligned}} \tag{26}$$

When computing the optimal rotation matrix, $\mathbf{R}$ must satisfy the SO(3) constraint, i.e., its determinant must be $+1$: $\det(\mathbf{R}) = 1$. However, in practice one may obtain $\det(\mathbf{R}) = -1$, which corresponds to a reflection (a flip about some axis). To prevent this degenerate case, the optimal rotation is commonly computed as follows.

$$\mathbf{R} = \mathbf{V}\begin{bmatrix}1 & & \\ & 1 & \\ & & \det(\mathbf{V}\mathbf{U}^\intercal)\end{bmatrix}\mathbf{U}^\intercal \tag{27}$$

4.2 With unknown data associations

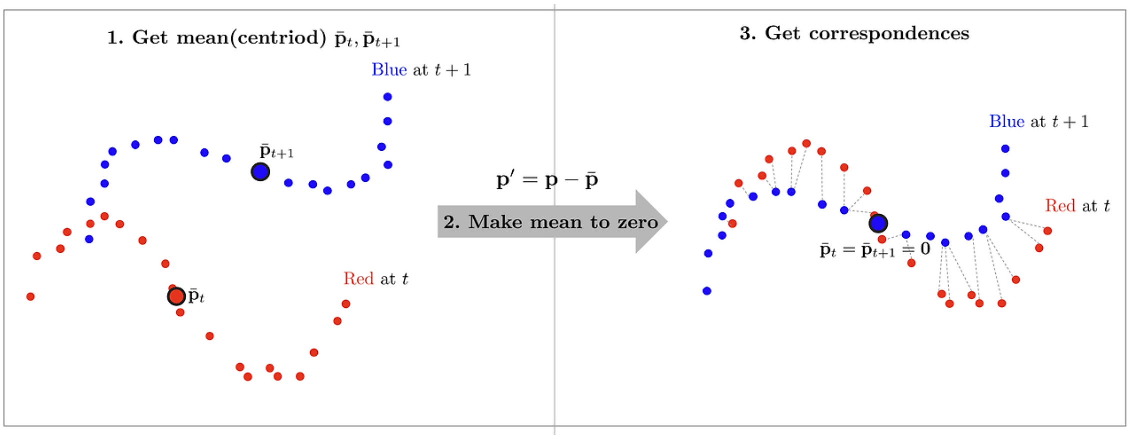
The optimal solution $\mathbf{R}^*, \mathbf{t}^*$ in the previous section applies when all correspondences (associations) between the two point clouds are known. In general, however, point-cloud data acquired from sensors does not tell us which point corresponds to which.

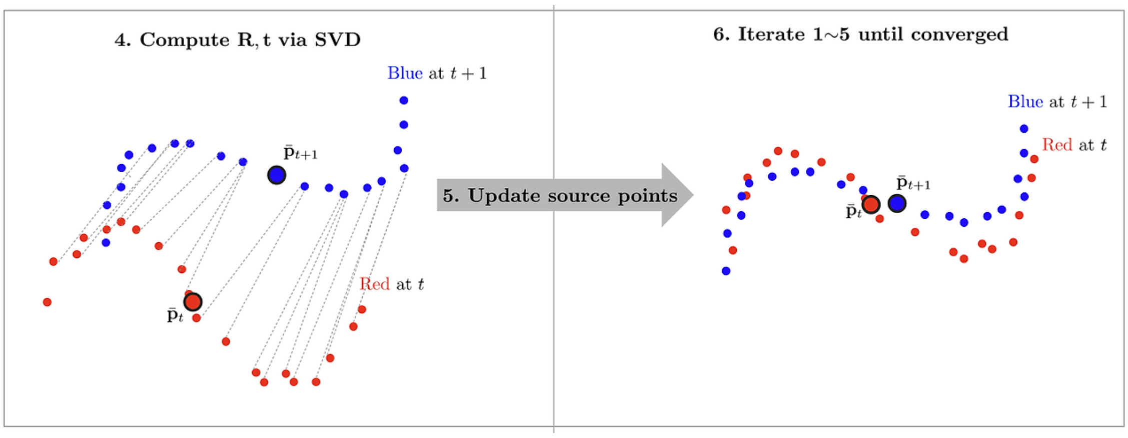


When correspondences are unknown as above, there is no closed-form solution available directly. **Iterative Closest Point (ICP) is an algorithm that iteratively estimates the optimum by assigning, for each point, its closest point as a correspondence pair.** The overall ICP procedure is as follows. (j: current iteration index)

1. Compute the mean (or centroid) of the source and target point clouds, $\bar{\mathbf{p}}_t, \bar{\mathbf{p}}_{t+1}$.
2. Normalize each point cloud to have zero mean by subtracting its centroid. ($\mathbf{p}' = \mathbf{p} - \bar{\mathbf{p}}$)
3. For each source point, set the nearest target point as its correspondence (using a nearest-neighbor method, e.g., KD-tree).
4. Compute the rotation $\mathbf{R}$ by SVD of the covariance matrix, and compute the translation vector $\mathbf{t}$ from the difference between the means. ($\mathbf{R}_j = \mathbf{V}\mathbf{U}^\intercal, \mathbf{t}_j = \bar{\mathbf{p}}_{t+1} - \mathbf{R}_j\bar{\mathbf{p}}_t$)
5. Transform the source point cloud by the estimated solution. $\mathbf{p}_{t,j+1} = \mathbf{R}_j\mathbf{p}_{t,j} + \mathbf{t}_j$
6. Repeat steps 1∼5 until the distance between the two point clouds becomes sufficiently small.

The ICP procedure can be illustrated as follows.


1. Get mean(centriod) $\bar{\mathbf{p}}_t, \bar{\mathbf{p}}_{t+1}$
Blue at $t+1$
$\bar{\mathbf{p}}_{t+1}$
$\bar{\mathbf{p}}_t$
Red at t
$\mathbf{p}' = \mathbf{p} - \bar{\mathbf{p}}$
2. Make mean to zero
3. Get correspondences
Blue at $t+1$
Red at t
$\bar{\mathbf{p}}_t = \bar{\mathbf{p}}_{t+1} = 0$



The algorithm described so far is commonly called **Vanila ICP** in the sense that it is the most basic form of ICP. Vanila ICP is relatively easy to implement and works well when the initial guess is accurate, but it generally requires many iterations to converge and is sensitive to incorrect correspondences, which can degrade the result.

To overcome the limitations of Vanila ICP, many variants have been proposed: running ICP on only a subset of points (e.g., keypoints), using different correspondence models such as point-to-plane, weighting correspondences to reduce the influence of outliers, or removing potential outliers altogether for more robust registration. For details, see Prof. Cyrill's ICP lecture.

4.3 Least squares point-to-point ICP (2D)

The key idea in the ICP method described so far was to compute the covariance matrix of the two point clouds and then obtain the rotation matrix as $\mathbf{R} = \mathbf{V}\mathbf{U}^\intercal$ via SVD. In this section, we explain how to solve the ICP problem using nonlinear least squares (= Gauss–Newton, GN). Least-squares ICP can be applied when correspondences are unknown (unknown data association).

Least-squares ICP is identical to the conventional ICP in most steps, but it differs in how the per-iteration optimum $\mathbf{R}, \mathbf{t}$ is estimated: SVD is replaced by the Gauss–Newton method. It also differs in that, when finding the closest correspondence pairs, it finds correspondences directly without first centering the point clouds to have zero mean.

The SVD solution assumes point-to-point correspondences between the point clouds. In practice, ICP can use various error functions beyond point-to-point; with least-squares ICP, such error functions can be optimized in a consistent way. One can also include terms such as robust estimators to perform optimization that is more resistant to outliers.

To perform least-squares ICP, define the 2D pose state vector $\mathbf{x} = [t_x, t_y, \theta]^\intercal$. The 2D rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ are as follows.

$$\begin{aligned}
\mathbf{x} &= [t_x, t_y, \theta]^\intercal \\
\mathbf{R}(\theta) &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\
\mathbf{t} &= \begin{bmatrix} t_x \\ t_y \end{bmatrix}
\end{aligned} \tag{28}$$

- $\mathbf{R}(\theta)$: explicit notation indicating that the rotation matrix takes θ as an input parameter.

The optimization is the same as (2), but for convenience in deriving the Jacobian we change the residual from the 'measurement - prediction' form to the 'prediction - measurement' form. The error function $\mathbf{e}_i$ for the i-th point can be written as follows.

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{R}(\theta)\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i} \in \mathbb{R}^2 \tag{29}$$

$$\boxed{\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} \|\mathbf{e}_i(\mathbf{x})\|^2 \\ &= \arg\min_{\mathbf{x}} \|\mathbf{R}(\theta)\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i}\|^2 \end{aligned}} \tag{30}$$

- Even if the residual is changed to the 'prediction - measurement' form, the overall optimization is unaffected because it is squared. However, the sign of the Jacobian depends on the residual definition, so be careful in implementation.

Since the above is a nonlinear least-squares form, it can be solved by the Gauss–Newton (GN) or Levenberg–Marquardt (LM) method. As an example, consider GN. For a small increment $\Delta\mathbf{x}$, the error function $\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})$ can be linearized via a Taylor approximation as follows.

$$\mathbf{e}_i(\mathbf{x}+\Delta\mathbf{x}) \approx \mathbf{e}_i(\mathbf{x}) + \mathbf{J}_i\Delta\mathbf{x} \tag{31}$$

The Jacobian $\mathbf{J}_i$ for the i-th point is as follows.

$$\boxed{\begin{aligned} \mathbf{J}_i &= \frac{\partial \mathbf{e}_i}{\partial \mathbf{x}} \\ &= \begin{bmatrix} \frac{\partial \mathbf{e}_i}{\partial t_x} & \frac{\partial \mathbf{e}_i}{\partial t_y} & \frac{\partial \mathbf{e}_i}{\partial \theta} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial \mathbf{e}_i^x}{\partial t_x} & \frac{\partial \mathbf{e}_i^x}{\partial t_y} & \frac{\partial \mathbf{e}_i^x}{\partial \theta} \\ \frac{\partial \mathbf{e}_i^y}{\partial t_x} & \frac{\partial \mathbf{e}_i^y}{\partial t_y} & \frac{\partial \mathbf{e}_i^y}{\partial \theta} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -\sin\theta\; x_{t,i} - \cos\theta\; y_{t,i} \\ 0 & 1 & \cos\theta\; x_{t,i} - \sin\theta\; y_{t,i} \end{bmatrix} \in \mathbb{R}^{2\times 3} \end{aligned}} \tag{32}$$

In the above, the Jacobian component with respect to θ can be derived as follows.

$$\boxed{\begin{aligned} \frac{\partial \mathbf{e}_i}{\partial \theta} &= \frac{\partial}{\partial \theta}\Big(\mathbf{R}(\theta)\mathbf{p}_{t,i}\Big) \\ &= \frac{\partial}{\partial \theta}\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x_{t,i} \\ y_{t,i} \end{bmatrix} \\ &= \begin{bmatrix} -\sin\theta & -\cos\theta \\ \cos\theta & -\sin\theta \end{bmatrix}\begin{bmatrix} x_{t,i} \\ y_{t,i} \end{bmatrix} \\ &= \begin{bmatrix} -\sin\theta\; x_{t,i} - \cos\theta\; y_{t,i} \\ \cos\theta\; x_{t,i} - \sin\theta\; y_{t,i} \end{bmatrix} \in \mathbb{R}^{2\times 1} \end{aligned}} \tag{33}$$

- $\mathbf{p}_{t,i} = [x_{t,i}, y_{t,i}]^\intercal$

Combining the error functions over all points yields the point-cloud error function as follows.

$$\begin{aligned} \mathbf{E}(\mathbf{x}) &= \sum_i^n \|\mathbf{e}_i(\mathbf{x})\|^2 \\ \mathbf{x} &= \arg\min_{\mathbf{x}} \mathbf{E}(\mathbf{x}) \end{aligned} \tag{34}$$

The point-cloud Jacobian $\mathbf{J}$ and the corresponding $\mathbf{H}, \mathbf{b}$ are aggregated as follows.

$$\begin{aligned} \mathbf{H}_i &= \mathbf{J}_i^\intercal \mathbf{J}_i \\ \mathbf{b}_i &= \mathbf{J}_i^\intercal \mathbf{e}_i \end{aligned} \tag{35}$$

$$\begin{aligned} \mathbf{J} &= \sum_i^n \mathbf{J}_i \\ \mathbf{H} &= \sum_i^n \mathbf{H}_i \\ \mathbf{b} &= \sum_i^n \mathbf{b}_i \end{aligned} \tag{36}$$

The GN solution can be obtained as follows. For readers interested in the derivation, see the post on error and Jacobian.

$$\Delta \mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b} \tag{37}$$

After computing the optimal increment $\Delta \mathbf{x}^*$ as above, update the original state variable $\mathbf{x}$. Through this update, the source point cloud is gradually registered to the target point cloud.

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}^* \tag{38}$$

Repeat the above process until the source point cloud is no longer updated. This procedure is called the Least-squares ICP (2D ver.) algorithm.

Gauss-Newton method (point-to-point ICP 2D)

1. Set correspondences by assigning to each source point the closest target point using a nearest-neighbor method (e.g., KD-tree).
2. Define the error function as in (29). $\mathbf{e}(\mathbf{x})$
3. Linearize via Taylor expansion and compute the Jacobian. $\mathbf{H} = \mathbf{J}^\intercal\mathbf{J}, \mathbf{b} = \mathbf{J}^\intercal\mathbf{e}$
4. Set the first derivative to zero. $\Delta \mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$
5. Compute the increment and apply it to the state. $\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}^*$
6. Repeat until convergence.

4.3.1 Least squares point-to-point ICP (3D)

Least-squares ICP for 3D point clouds is identical to the 2D case except for the Jacobian. Note that **because the Jacobian includes the 3D rotation matrix $\mathbf{R} \in SO(3)$, Lie algebra $so(3)$-based optimization is used when computing the Jacobian.** For Lie-theory-based optimization, see the post on error and Jacobian.

To perform least-squares ICP, define the 3D pose state variable $\mathbf{x} = [t_x, t_y, t_z, \mathbf{R}]^\intercal$. The 3D rotation matrix $\mathbf{R} \in SO(3)$ and translation vector $\mathbf{t}$ are as follows.

$$\begin{aligned} \mathbf{x} &= [t_x, t_y, t_z, \mathbf{R}]^\intercal \\ \mathbf{R} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in SO(3) \\ \mathbf{t} &= \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \end{aligned} \tag{39}$$

The error function $\mathbf{e}_i$ for the i-th point can be written as follows.

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i} \in \mathbb{R}^3 \tag{40}$$

$$\boxed{\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} \|\mathbf{e}_i(\mathbf{x})\|^2 \\ &= \arg\min_{\mathbf{x}} \|\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i}\|^2 \end{aligned}} \tag{41}$$

Since the above is a nonlinear least-squares form, it can be solved by the Gauss–Newton (GN) or Levenberg–Marquardt (LM) method. As an example, consider GN. For a small increment $\Delta\mathbf{x}$, the error function $\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})$ can be linearized via a Taylor approximation as follows.

$$\mathbf{e}_i(\mathbf{x}+\Delta\mathbf{x}) \approx \mathbf{e}_i(\mathbf{x}) + \mathbf{J}_i\Delta\mathbf{x} \tag{42}$$

The Jacobian $\mathbf{J}_i$ for the i-th 3D point is as follows.

$$\boxed{\begin{aligned}\mathbf{J}_i &= \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial[\mathbf{t},\mathbf{R}]} \\ &= \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial[\mathbf{t},\Delta\mathbf{w}]} \quad \leftarrow \text{so(3)-based optimization} \\ &= \frac{\partial}{\partial[\mathbf{t},\Delta\mathbf{w}]}\bigg(\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i}\bigg) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & z'_{t,i} & -y'_{t,i} \\ 0 & 1 & 0 & -z'_{t,i} & 0 & x'_{t,i} \\ 0 & 0 & 1 & y'_{t,i} & -x'_{t,i} & 0 \end{bmatrix} \in \mathbb{R}^{3\times 6}\end{aligned}} \tag{43}$$

- For Lie-theory-based optimization, see the post on error and Jacobian.

$\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{t}}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{t}} &= \frac{\partial}{\partial \mathbf{t}}\mathbf{t} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}\end{aligned}} \tag{44}$$

$\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \Delta\mathbf{w}}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \Delta\mathbf{w}} &= -[\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t}]_\times \\ &= \begin{bmatrix} 0 & z'_{t,i} & -y'_{t,i} \\ -z'_{t,i} & 0 & x'_{t,i} \\ y'_{t,i} & -x'_{t,i} & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}\end{aligned}} \tag{45}$$

- $\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} = [x'_{t,i}\ y'_{t,i}\ z'_{t,i}]^\intercal$
- $[\mathbf{a}]_\times = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_\times = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$: skew-symmetric matrix operator

Combining the error functions over all points yields the point-cloud error function as follows.

$$\begin{aligned}\mathbf{E}(\mathbf{x}) &= \sum_i^n \|\mathbf{e}_i(\mathbf{x})\|^2 \\ \mathbf{x} &= \arg\min_{\mathbf{x}} \mathbf{E}(\mathbf{x})\end{aligned} \tag{46}$$

The point-cloud Jacobian $\mathbf{J}$ and the corresponding $\mathbf{H}, \mathbf{b}$ are aggregated as follows.

$$\begin{aligned}\mathbf{H}_i &= \mathbf{J}_i^\intercal \mathbf{J}_i \\ \mathbf{b}_i &= \mathbf{J}_i^\intercal \mathbf{e}_i\end{aligned} \tag{47}$$

$$
\begin{aligned}
\mathbf{J} &= \sum_i^n \mathbf{J}_i \\
\mathbf{H} &= \sum_i^n \mathbf{H}_i \\
\mathbf{b} &= \sum_i^n \mathbf{b}_i
\end{aligned} \tag{48}
$$

The GN solution can be obtained as follows. For readers interested in the derivation, see the post on error and Jacobian.

$$\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b} \tag{49}$$

After computing the optimal increment $\Delta\mathbf{x}^*$ as above, update the original state variable $\mathbf{x}$. Through this update, the source point cloud is gradually registered to the target point cloud.

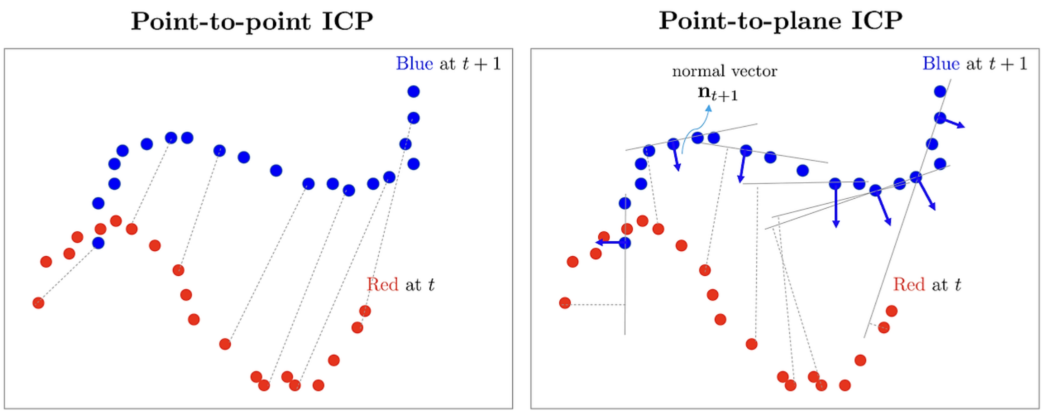
$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^* \tag{50}$$

Repeat the above process until the source point cloud is no longer updated. This procedure is called the Least-squares ICP (3D ver.) algorithm.

Gauss-Newton method (point-to-point ICP 3D)

1. Set correspondences by assigning to each source point the closest target point using a nearest-neighbor method (e.g., KD-tree).
2. Define the error function as in (40). $\mathbf{e}(\mathbf{x})$
3. Linearize via Taylor expansion and compute the Jacobian. $\mathbf{H} = \mathbf{J}^\intercal\mathbf{J}, \mathbf{b} = \mathbf{J}^\intercal\mathbf{e}$
4. Set the first derivative to zero. $\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$
5. Compute the increment and apply it to the state. $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^*$
6. Repeat until convergence.

5 Point-to-plane ICP



Conventional point-to-point ICP performs optimization to minimize the Euclidean distance between a source point and a target point. In contrast, point-to-plane ICP performs optimization to minimize the distance between the source point and the target plane along the target normal direction. Foundational papers for point-to-plane ICP include [5], [6], and [7].

Compared to point-to-point, point-to-plane ICP typically has **faster convergence and is less sensitive to noise and outliers**. On the other hand, there is a trade-off: **the computational cost increases** because one must estimate normals and then optimize. Also, since the optimization formulation differs from point-to-point, the SVD solution is not applicable and the optimum can only be found via least squares (= GN).

5.1 Least squares point-to-plane ICP (2D)

Most steps of point-to-plane ICP are identical to point-to-point ICP, but it differs in that it computes normal vectors and slightly modifies the optimization formulation. To perform least-squares ICP, define the 2D pose state variable $\mathbf{x} = [t_x, t_y, \theta]^\intercal$. The 2D rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ are as follows.

$$\begin{aligned} \mathbf{x} &= [t_x, t_y, \theta]^\intercal \\ \mathbf{R}(\theta) &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ \mathbf{t} &= \begin{bmatrix} t_x \\ t_y \end{bmatrix} \end{aligned} \tag{51}$$

- $\mathbf{R}(\theta)$: explicit notation indicating that the rotation matrix takes θ as an input parameter.

The error function $\mathbf{e}_i$ for the i-th point can be written as follows.

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{n}_{t+1,i}^\intercal(\mathbf{R}(\theta)\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i}) \in \mathbb{R} \tag{52}$$

$$\boxed{\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} \|\mathbf{e}_i(\mathbf{x})\|^2 \\ &= \arg\min_{\mathbf{x}} \|\mathbf{n}_{t+1,i}^\intercal(\mathbf{R}(\theta)\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i})\|^2 \end{aligned}} \tag{53}$$

- $\mathbf{n}_{t+1,i} = [n_{t+1,i}^x, n_{t+1,i}^y]^\intercal$: the normal vector of the i-th point in the target point cloud
- Because of the identity $\mathbf{n}^\intercal(\cdot) = (\cdot)^\intercal\mathbf{n}$, the normal vector can appear on either the right or left side of the expression.

We can see that the formulation is constructed by taking the dot product of the normal vector $\mathbf{n}_{t+1,i}$ with the conventional point-to-point objective. **This can be interpreted as using the fact that, when the source point and target point are perfectly registered, the vector between them is orthogonal to the normal vector, hence the dot product becomes 0.**

In 2D, when a point $\mathbf{p} = [x, y]^\intercal$ is given, a normal vector can be obtained simply as $\mathbf{n} = [-y, x]^\intercal$.

Since the above is a nonlinear least-squares form, it can be solved by the Gauss–Newton (GN) or Levenberg–Marquardt (LM) method. As an example, consider GN. For a small increment $\Delta\mathbf{x}$, the error function $\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})$ can be linearized via a Taylor approximation as follows.

$$\mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) \approx \mathbf{e}_i(\mathbf{x}) + \mathbf{J}_i\Delta\mathbf{x} \tag{54}$$

The Jacobian $\mathbf{J}_i$ for the i-th point is as follows.

$$\boxed{\begin{aligned} \mathbf{J}_i &= \frac{\partial \mathbf{e}_i}{\partial \mathbf{x}} \\ &= \begin{bmatrix} \frac{\partial \mathbf{e}_i}{\partial t_x} & \frac{\partial \mathbf{e}_i}{\partial t_y} & \frac{\partial \mathbf{e}_i}{\partial \theta} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial \mathbf{e}_i^x}{\partial t_x} & \frac{\partial \mathbf{e}_i^x}{\partial t_y} & \frac{\partial \mathbf{e}_i^x}{\partial \theta} \\ \frac{\partial \mathbf{e}_i^y}{\partial t_x} & \frac{\partial \mathbf{e}_i^y}{\partial t_y} & \frac{\partial \mathbf{e}_i^y}{\partial \theta} \end{bmatrix} \\ &= \begin{bmatrix} n_{t+1,i}^x & n_{t+1,i}^y & n_{t+1,i}^x(-\sin\theta\ x_{t,i} - \cos\theta\ y_{t,i}) + n_{t+1,i}^y(\cos\theta\ x_{t,i} - \sin\theta\ y_{t,i}) \end{bmatrix} \in \mathbb{R}^{1\times 3} \end{aligned}} \tag{55}$$

In the above, the Jacobian component with respect to θ can be derived as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{e}_i}{\partial \theta} &= \frac{\partial}{\partial \theta}\Big(\mathbf{n}_{t+1,i}^\intercal \mathbf{R}(\theta)\mathbf{p}_{t,i}\Big) \\ &= \frac{\partial}{\partial \theta}\begin{bmatrix} n_{t+1,i}^x & n_{t+1,i}^y \end{bmatrix}\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x_{t,i} \\ y_{t,i} \end{bmatrix} \\ &= \begin{bmatrix} n_{t+1,i}^x & n_{t+1,i}^y \end{bmatrix}\begin{bmatrix} -\sin\theta & -\cos\theta \\ \cos\theta & -\sin\theta \end{bmatrix}\begin{bmatrix} x_{t,i} \\ y_{t,i} \end{bmatrix} \\ &= \Big[n_{t+1,i}^x(-\sin\theta\ x_{t,i} - \cos\theta\ y_{t,i}) - n_{t+1,i}^y(\cos\theta\ x_{t,i} - \sin\theta\ y_{t,i})\Big] \in \mathbb{R}\end{aligned}} \tag{56}$$

- $\mathbf{p}_{t,i} = [x_{t,i}, y_{t,i}]^\intercal$
- $\mathbf{n}_{t+1,i} = [n_{t+1,i}^x, n_{t+1,i}^y]^\intercal$

Combining the error functions over all points yields the point-cloud error function as follows.

$$\begin{aligned}\mathbf{E}(\mathbf{x}) &= \sum_i^n \|\mathbf{e}_i(\mathbf{x})\|^2 \\ \mathbf{x} &= \arg\min_{\mathbf{x}} \mathbf{E}(\mathbf{x})\end{aligned} \tag{57}$$

The point-cloud Jacobian $\mathbf{J}$ and the corresponding $\mathbf{H}, \mathbf{b}$ are aggregated as follows.

$$\begin{aligned}\mathbf{H}_i &= \mathbf{J}_i^\intercal \mathbf{J}_i \\ \mathbf{b}_i &= \mathbf{J}_i^\intercal \mathbf{e}_i\end{aligned} \tag{58}$$

$$\begin{aligned}\mathbf{J} &= \sum_i^n \mathbf{J}_i \\ \mathbf{H} &= \sum_i^n \mathbf{H}_i \\ \mathbf{b} &= \sum_i^n \mathbf{b}_i\end{aligned} \tag{59}$$

The GN solution can be obtained as follows. For readers interested in the derivation, see the post on error and Jacobian.

$$\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b} \tag{60}$$

After computing the optimal increment $\Delta\mathbf{x}^*$ as above, update the original state variable $\mathbf{x}$. Through this update, the source point cloud is gradually registered to the target point cloud.

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^* \tag{61}$$

Repeat the above process until the source point cloud is no longer updated. This procedure is called the Least-squares ICP (2D ver.) algorithm.

Gauss-Newton method (point-to-plane ICP 2D)

1. Set correspondences by assigning to each source point the closest target point using a nearest-neighbor method (e.g., KD-tree).
2. Compute the normal vectors $\mathbf{n}$ of the target points. (2D: $\mathbf{p} = [x, y]^\intercal, \mathbf{n} = [-y, x]^\intercal$)
3. Define the error function as in (52). $\mathbf{e}(\mathbf{x})$
4. Linearize via Taylor expansion and compute the Jacobian. $\mathbf{H} = \mathbf{J}^\intercal\mathbf{J}, \mathbf{b} = \mathbf{J}^\intercal\mathbf{e}$
5. Set the first derivative to zero. $\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$

6. Compute the increment and apply it to the state. $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^*$
7. Repeat until convergence.

5.2 Least squares point-to-plane ICP (3D)

In 3D as well, most steps of point-to-plane ICP are identical to point-to-point ICP, but it differs in that it computes normal vectors and slightly modifies the optimization formulation. To perform least-squares ICP, define the 3D pose state variable $\mathbf{x} = [t_x, t_y, t_z, \mathbf{R}]^\intercal$. The 3D rotation matrix $\mathbf{R} \in SO(3)$ and translation vector $\mathbf{t}$ are as follows.

$$\begin{aligned} \mathbf{x} &= [t_x, t_y, t_z, \mathbf{R}]^\intercal \\ \mathbf{R} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in SO(3) \\ \mathbf{t} &= \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \end{aligned} \tag{62}$$

The error function $\mathbf{e}_i$ for the i-th point can be written as follows.

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{n}_{t+1,i}^\intercal(\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i}) \in \mathbb{R} \tag{63}$$

$$\boxed{\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} \|\mathbf{e}_i(\mathbf{x})\|^2 \\ &= \arg\min_{\mathbf{x}} \|\mathbf{n}_{t+1,i}^\intercal(\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i})\|^2 \end{aligned}} \tag{64}$$

- $\mathbf{n}_{t+1,i} = [n_{t+1,i}^x, n_{t+1,i}^y, n_{t+1,i}^z]^\intercal$: the normal vector of the i-th point in the target point cloud
- Because of the identity $\mathbf{n}^\intercal(\cdot) = (\cdot)^\intercal\mathbf{n}$, the normal vector can appear on either the right or left side of the expression.

We can see that the formulation is constructed by taking the dot product of the normal vector $\mathbf{n}_{t+1,i}$ with the conventional point-to-point objective. **This can be interpreted as using the fact that, when the source point and target point are perfectly registered, the vector between them is orthogonal to the normal vector, hence the dot product becomes 0.**

Normal vectors can be computed in various ways. In 3D, given a point $\mathbf{p} = [x, y, z]^\intercal$, one can find the two nearest neighbors $\mathbf{p}_1, \mathbf{p}_2$ (e.g., via KD-tree) and take the cross product $\mathbf{n} = \mathbf{p}_1 \times \mathbf{p}_2$. Alternatively, one can find the k neighboring points and use the fact that, in PCA, the eigenvector corresponding to the smallest eigenvalue is the normal vector.

Since the above is a nonlinear least-squares form, it can be solved by the Gauss–Newton (GN) or Levenberg–Marquardt (LM) method. As an example, consider GN. For a small increment $\Delta\mathbf{x}$, the error function $\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})$ can be linearized via a Taylor approximation as follows.

$$\mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) \approx \mathbf{e}_i(\mathbf{x}) + \mathbf{J}_i\Delta\mathbf{x} \tag{65}$$

The Jacobian $\mathbf{J}_i$ for the i-th 3D point is as follows.

$$\boxed{\begin{aligned}\mathbf{J}_i &= \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial [\mathbf{t}, \mathbf{R}]} \\ &= \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial [\mathbf{t}, \Delta\mathbf{w}]} \quad \leftarrow \text{so(3)-based optimization} \\ &= \frac{\partial}{\partial [\mathbf{t}, \Delta\mathbf{w}]} \bigg(\mathbf{n}_{t+1,i}^{\intercal} (\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i}) \bigg) \\ &= \begin{bmatrix} n_{t+1,i}^{x} & n_{t+1,i}^{y} & n_{t+1,i}^{z} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & z'_{t,i} & -y'_{t,i} \\ 0 & 1 & 0 & -z'_{t,i} & 0 & x'_{t,i} \\ 0 & 0 & 1 & y'_{t,i} & -x'_{t,i} & 0 \end{bmatrix} \\ &= \begin{bmatrix} n_{t+1,i}^{x} & n_{t+1,i}^{y} & n_{t+1,i}^{z} & n_{t+1,i}^{x}(-z'_{t,i} + y'_{t,i}) & n_{t+1,i}^{y}(z'_{t,i} - x'_{t,i}) & n_{t+1,i}^{z}(-y'_{t,i} + x'_{t,i}) \end{bmatrix} \in \mathbb{R}^{1\times 6}\end{aligned}} \tag{66}$$

- For Lie-theory-based optimization, see the post on error and Jacobian.

$\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{t}}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{t}} &= \frac{\partial}{\partial \mathbf{t}} \mathbf{n}_{t+1,i}^{\intercal} \mathbf{t} \\ &= \begin{bmatrix} n_{t+1,i}^{x} & n_{t+1,i}^{y} & n_{t+1,i}^{z} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} n_{t+1,i}^{x} & n_{t+1,i}^{y} & n_{t+1,i}^{z} \end{bmatrix} \in \mathbb{R}^{1\times 3}\end{aligned}} \tag{67}$$

$\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \Delta\mathbf{w}}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \Delta\mathbf{w}} &= -\mathbf{n}_{t+1,i}^{\intercal} [\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t}]_{\times} \\ &= \begin{bmatrix} n_{t+1,i}^{x} & n_{t+1,i}^{y} & n_{t+1,i}^{z} \end{bmatrix} \begin{bmatrix} 0 & z'_{t,i} & -y'_{t,i} \\ -z'_{t,i} & 0 & x'_{t,i} \\ y'_{t,i} & -x'_{t,i} & 0 \end{bmatrix} \\ &= \begin{bmatrix} n_{t+1,i}^{x}(-z'_{t,i} + y'_{t,i}) & n_{t+1,i}^{y}(z'_{t,i} - x'_{t,i}) & n_{t+1,i}^{z}(-y'_{t,i} + x'_{t,i}) \end{bmatrix} \in \mathbb{R}^{1\times 3}\end{aligned}} \tag{68}$$

- $\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} = [x'_{t,i} \ y'_{t,i} \ z'_{t,i}]^{\intercal}$
- $[\mathbf{a}]_{\times} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_{\times} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$: skew-symmetric matrix operator

Combining the error functions over all points yields the point-cloud error function as follows.

$$\begin{aligned}\mathbf{E}(\mathbf{x}) &= \sum_{i}^{n} \|\mathbf{e}_i(\mathbf{x})\|^2 \\ \mathbf{x} &= \arg\min_{\mathbf{x}} \mathbf{E}(\mathbf{x})\end{aligned} \tag{69}$$

The point-cloud Jacobian $\mathbf{J}$ and the corresponding $\mathbf{H}, \mathbf{b}$ are aggregated as follows.

$$\begin{aligned}\mathbf{H}_i &= \mathbf{J}_i^{\intercal} \mathbf{J}_i \\ \mathbf{b}_i &= \mathbf{J}_i^{\intercal} \mathbf{e}_i\end{aligned} \tag{70}$$

$$\mathbf{J} = \sum_i^n \mathbf{J}_i$$
$$\mathbf{H} = \sum_i^n \mathbf{H}_i \tag{71}$$
$$\mathbf{b} = \sum_i^n \mathbf{b}_i$$

The GN solution can be obtained as follows. For readers interested in the derivation, see the post on error and Jacobian.

$$\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b} \tag{72}$$

After computing the optimal increment $\Delta\mathbf{x}^*$ as above, update the original state variable $\mathbf{x}$. Through this update, the source point cloud is gradually registered to the target point cloud.

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^* \tag{73}$$

Repeat the above process until the source point cloud is no longer updated. This procedure is called the Least-squares ICP (3D ver.) algorithm.

Gauss-Newton method (point-to-plane ICP 3D)

1. Set correspondences by assigning to each source point the closest target point using a nearest-neighbor method (e.g., KD-tree).
2. Compute the normal vectors $\mathbf{n}$ of the target points.
3. Define the error function as in (63). $\mathbf{e}(\mathbf{x})$
4. Linearize via Taylor expansion and compute the Jacobian. $\mathbf{H} = \mathbf{J}^\intercal\mathbf{J}, \mathbf{b} = \mathbf{J}^\intercal\mathbf{e}$
5. Set the first derivative to zero. $\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$
6. Compute the increment and apply it to the state. $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^*$
7. Repeat until convergence.

6 Generalized-ICP (GICP)

Generalized-ICP (GICP) is an algorithm that, unlike conventional ICP variants, **models points probabilistically and estimates the transformation between point clouds**. Depending on the form of the covariance matrices, GICP can encompass point-to-point, point-to-plane, and plane-to-plane ICP. This is presumably why it is called a generalized ICP. When finding closest correspondences, however, it still uses distance-based nearest neighbors (e.g., KD-tree) rather than probabilistic association, preserving the speed of KNN search.

GICP[8] derives its formulation assuming that correspondence pairs are given. When the source point cloud is $\mathbf{p}_t = [\mathbf{p}_{t,1}, \mathbf{p}_{t,2}, \cdots, \mathbf{p}_{t,n}]^\intercal$ and the target point cloud is $\mathbf{p}_{t+1} = [\mathbf{p}_{t+1,1}, \mathbf{p}_{t+1,2}, \cdots, \mathbf{p}_{t+1,n}]^\intercal$, **each point is modeled as following a Gaussian distribution.**

$$\begin{aligned} \mathbf{p}_{t,i} &\sim \mathcal{N}(\hat{\mathbf{p}}_{t,i}, \mathbf{C}_{t,i}) \\ \mathbf{p}_{t+1,i} &\sim \mathcal{N}(\hat{\mathbf{p}}_{t+1,i}, \mathbf{C}_{t+1,i}) \end{aligned} \tag{74}$$

$$\begin{aligned} \hat{\mathbf{p}}_t &= [\hat{\mathbf{p}}_{t,1}, \hat{\mathbf{p}}_{t,2}, \cdots, \hat{\mathbf{p}}_{t,n}]^\intercal \\ \hat{\mathbf{p}}_{t+1} &= [\hat{\mathbf{p}}_{t+1,1}, \hat{\mathbf{p}}_{t+1,2}, \cdots, \hat{\mathbf{p}}_{t+1,n}]^\intercal \end{aligned} \tag{75}$$

If the two point clouds are separated exactly by a Euclidean distance without noise or outliers, the following transformation relationship holds between the two points.

$$\hat{\mathbf{p}}_{t+1,i} = \mathbf{T}^* \hat{\mathbf{p}}_{t,i} \tag{76}$$

For an arbitrary transformation $\mathbf{T}$ between two points,

$$\boxed{d_i = \hat{\mathbf{p}}_{t+1,i} - \mathbf{T}\hat{\mathbf{p}}_{t,i}} \tag{77}$$

we can define the distance function d_i as follows. Since both $\hat{\mathbf{p}}_{t,i}$ and $\hat{\mathbf{p}}_{t+1,i}$ are random variables, d_i also follows a probability distribution.

$$\begin{aligned} d_i|_{\mathbf{T}=\mathbf{T}^*} &\sim \mathcal{N}(\hat{\mathbf{p}}_{t+1,i} - \mathbf{T}^*\hat{\mathbf{p}}_{t,i}, \mathbf{C}_{t+1,i} + \mathbf{T}^*\mathbf{C}_{t,i}\mathbf{T}^{*\intercal}) \\ &= \mathcal{N}(\mathbf{0}, \mathbf{C}_{t+1,i} + \mathbf{T}^*\mathbf{C}_{t,i}\mathbf{T}^{*\intercal}) \end{aligned} \tag{78}$$

- Assume the two points $\hat{\mathbf{p}}_{t,i}, \hat{\mathbf{p}}_{t+1,i}$ are independent Gaussian distributions.

Tip

Linear transformation of gaussian random variable
If a random variable $\mathbf{x}$ follows a Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\mathbf{a}, \mathbf{B})$ with mean $\mathbf{a}$ and covariance $\mathbf{B}$, then for an arbitrary matrix $\mathbf{C}$, the variable $\mathbf{y} = \mathbf{Cx}$ is also random and follows $\mathbf{y} \sim \mathcal{N}(\mathbf{Ca}, \mathbf{CBC}^\intercal)$. For more details, see the probability theory post.

Since the above expression is the pdf for the distance between the i-th source and target points, summing over all point pairs and multiplying all pdfs $p(d_i)$ yields the following maximum-likelihood estimation (MLE) objective for $\mathbf{T}$.

$$\boxed{\begin{aligned} \mathbf{T} &= \arg\max_{\mathbf{T}} \Pi_i p(d_i) \quad \leftarrow \text{likelihood} \\ &= \arg\max_{\mathbf{T}} \sum_i \log p(d_i) \quad \leftarrow \text{log-likelihood} \end{aligned}} \tag{79}$$

$p(d_i)$ is as follows. **Its mean is omitted because it becomes 0, e.g., $d_i|_{\mathbf{T}=\mathbf{T}^*} = 0$.**

$$\begin{aligned} p(d_i) &= \eta \cdot \exp\left(-\frac{1}{2} d_i^\intercal (\mathbf{C}_{t+1,i} + \mathbf{T}\mathbf{C}_{t,i}\mathbf{T}^\intercal)^{-1} d_i \right) \\ &\sim \mathcal{N}(\mathbf{0}, \mathbf{C}_{t+1,i} + \mathbf{T}\mathbf{C}_{t,i}\mathbf{T}^\intercal) \end{aligned} \tag{80}$$

Therefore, (79) can be rewritten as follows. In the log-likelihood, the $(-)$ sign is removed and the remaining terms convert argmax to argmin ($=$ negative log-likelihood).

$$\boxed{\mathbf{T} = \arg\min_{\mathbf{T}} \sum_i d_i^\intercal (\mathbf{C}_{t+1,i} + \mathbf{T}\mathbf{C}_{t,i}\mathbf{T}^\intercal)^{-1} d_i} \tag{81}$$

6.1 Point-to-point ICP in GICP

GICP is called a generalized ICP because, in (81), by choosing different forms for the covariance matrices $\mathbf{C}_{t,i}, \mathbf{C}_{t+1,i}$, one can derive the formulations of point-to-point, point-to-plane, and plane-to-plane ICP.

If the covariance matrices are as follows, the **point-to-point ICP** formulation is derived.

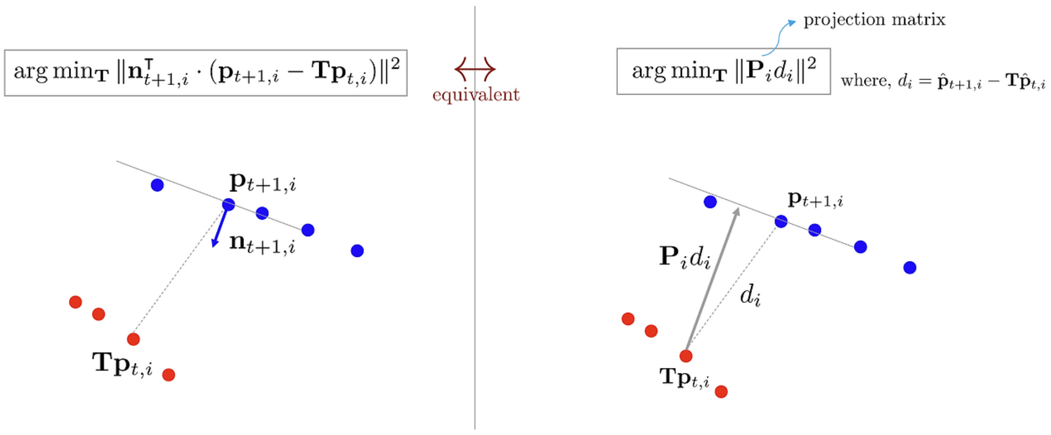
$$\begin{aligned} \mathbf{C}_{t+1,i} &= \mathbf{I} \\ \mathbf{C}_{t,i} &= \mathbf{0} \end{aligned} \tag{82}$$

Substituting the above into (81) yields the following.

$$\boxed{\begin{aligned}\mathbf{T} &= \arg\min_{\mathbf{T}} \sum_i d_i^\intercal d_i \\ &= \arg\min_{\mathbf{T}} \sum_i \|d_i\|^2\end{aligned}} \tag{83}$$

This matches exactly the point-to-point ICP formulation.

6.2 Point-to-plane ICP in GICP


projection matrix
$\arg\min_{\mathbf{T}} \|\mathbf{n}_{t+1,i}^\intercal \cdot (\mathbf{p}_{t+1,i} - \mathbf{T}\mathbf{p}_{t,i})\|^2$
$\leftrightarrow$ equivalent
$\arg\min_{\mathbf{T}} \|\mathbf{P}_i d_i\|^2$
where, $d_i = \hat{\mathbf{p}}_{t+1,i} - \mathbf{T}\hat{\mathbf{p}}_{t,i}$
$\mathbf{p}_{t+1,i}$
$\mathbf{n}_{t+1,i}$
$\mathbf{T}\mathbf{p}_{t,i}$
$\mathbf{p}_{t+1,i}$
$\mathbf{P}_i d_i$
d_i
$\mathbf{T}\mathbf{p}_{t,i}$

The point-to-plane ICP described earlier minimizes the distance between a source point and a target plane. Thus its formulation takes the dot product with the target plane normal vector $\mathbf{n}_i$. In GICP, we take a slightly different view: we interpret it as optimizing the distance vector d_i projected onto the target plane normal direction by multiplying with a projection matrix $\mathbf{P}_i$.

$$\mathbf{T} = \arg\min_{\mathbf{T}} \sum_i \|\mathbf{P}_i d_i\|^2 \tag{84}$$

In the above, $\mathbf{P}_i$ denotes the projection matrix that projects onto the normal direction of the target plane from the source point. By properties of projection matrices, $\mathbf{P}_i = \mathbf{P}_i^2 = \mathbf{P}_i^\intercal$. Therefore the above can be transformed as follows.

$$\begin{aligned}\|\mathbf{P}_i d_i\|^2 &= (\mathbf{P}_i d_i)^\intercal(\mathbf{P}_i d_i) \\ &= d_i^\intercal \mathbf{P}_i d_i\end{aligned} \tag{85}$$

Accordingly, (84) can be written as follows.

$$\boxed{\begin{aligned}\mathbf{T} &= \arg\min_{\mathbf{T}} \sum_i \|\mathbf{P}_i d_i\|^2 \\ &= \arg\min_{\mathbf{T}} \sum_i \|d_i^\intercal \mathbf{P}_i d_i\|^2\end{aligned}} \tag{86}$$

This corresponds to the case in the GICP formulation (81) where the covariance matrices take the following forms.

$$\begin{aligned}\mathbf{C}_{t+1,i} &= \mathbf{P}_i^{-1} \\ \mathbf{C}_{t,i} &= \mathbf{0}\end{aligned} \tag{87}$$

Strictly speaking, because the projection matrix $\mathbf{P}_i$ is rank deficient, it is not full-rank and thus has no inverse. However, one can approximate $\mathbf{P}_i$ with an invertible matrix $\mathbf{Q}_i$. Assume $\mathbf{Q}_i$ is similar to $\mathbf{P}_i$ but full-rank (so that an inverse exists). In the GICP formulation, the closer $\mathbf{Q}_i$ is to $\mathbf{P}_i$, the more GICP converges to point-to-plane ICP.

Tip

Projection Matrix P
The projection matrix $\mathbf{P}_i$ is rank deficient because it projects a point (or vector) in a higher-dimensional vector space onto a lower-dimensional subspace. In the point-to-plane example, the source point $\mathbf{T}\mathbf{p}_{t,i}$ can be interpreted as being projected onto the subspace (plane) formed by the target point $\mathbf{p}_{t+1,i}$. For more details on projection matrices, see the linear algebra notes.

6.3 Plane-to-plane ICP in GICP

Finally, GICP can also perform **plane-to-plane ICP**, which minimizes the distance between a source plane and a target plane. One might think this can be done by representing $\mathbf{C}_{t,i}$ in (87) as a nonzero projection matrix $\mathbf{P}_i'^{-1}$ instead of $\mathbf{0}$, but then both projection matrices $\mathbf{P}_i, \mathbf{P}_i'$ are rank deficient and singular, so even with an approximation it is hard to expect high accuracy.

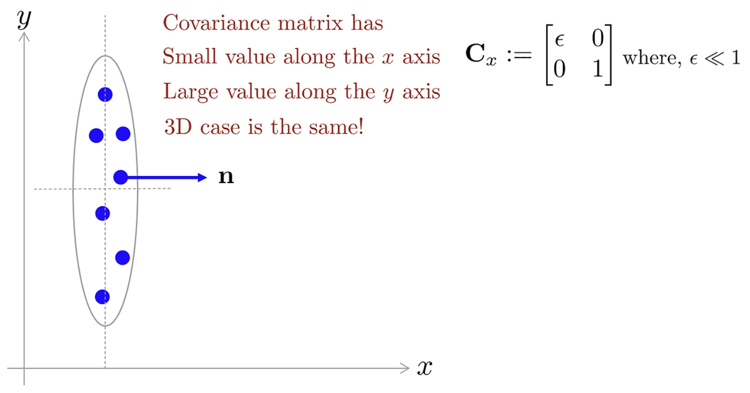
Accordingly, the plane covariance matrices are modeled under several assumptions.

1. Scan data samples a smooth 2-manifold in 3D space, so it is differentiable at every point (= normals can be computed).
2. Data sampled at different times $(t, t+1)$ generally does not sample exactly the same points. Therefore the nearest correspondence cannot have zero distance.
3. A sampled point is considered to have high covariance along the tangential (in-plane) direction and low covariance along the normal-vector direction.

If the normal vector is along the x-axis, then under the above assumptions the covariance $\mathbf{C}_x$ can be modeled as follows.

$$\mathbf{C}_x = \begin{bmatrix} \epsilon & & \\ & 1 & \\ & & 1 \end{bmatrix} \tag{88}$$

- ϵ: a small constant representing the covariance in the normal direction ($\epsilon \ll 1$)


y
Covariance matrix has
Small value along the x axis
Large value along the y axis
3D case is the same!
$\mathbf{C}_x := \begin{bmatrix} \epsilon & 0 \\ 0 & 1 \end{bmatrix}$ where, $\epsilon \ll 1$
$\mathbf{n}$
x

If we denote the normal vectors of the i-th source point $\mathbf{p}_{t,i}$ and target point $\mathbf{p}_{t+1,i}$ as μ_i and ν_i, respectively, then the plane-to-plane covariance matrices can be expressed as follows.

$$\begin{aligned} \mathbf{C}_{t,i} &= \mathbf{R}_{\mu_i} \begin{bmatrix} \epsilon & & \\ & 1 & \\ & & 1 \end{bmatrix} \mathbf{R}_{\mu_i}^{\intercal} \\ \mathbf{C}_{t+1,i} &= \mathbf{R}_{\nu_i} \begin{bmatrix} \epsilon & & \\ & 1 & \\ & & 1 \end{bmatrix} \mathbf{R}_{\nu_i}^{\intercal} \end{aligned} \tag{89}$$

- $\mathbf{R}_{\mu_i}$: a matrix that rotates the direction vector of the x-axis to μ_i
- $\mathbf{R}_{\nu_i}$: a matrix that rotates the direction vector of the x-axis to ν_i

By substituting the above into (81), plane-to-plane ICP can be performed. When two points with different normal directions are paired as correspondences, the summed covariance becomes isotropic and the contribution to the optimization becomes very small; that is, **it is robust to outliers**. However, since normals must be computed for all scan points, it also has **high computational cost**.

There are various ways to estimate normals. In the GICP paper, for each scanned point it finds the 20 nearest neighbors via KD-tree and then uses PCA on the covariance matrix to compute the normal vector. The eigenvector corresponding to the smallest eigenvalue is the normal vector.

6.4 Least squares GICP (3D)

GICP solves (81) by repeatedly applying nonlinear least squares (GN, LM). Compared to point-to-point and point-to-plane ICP described earlier, **almost all steps are the same except that the covariance from probabilistic modeling appears in the objective.** To perform least-squares GICP, define the 3D pose state variable $\mathbf{x} = [t_x, t_y, t_z, \mathbf{R}]^{\intercal}$. The 3D rotation matrix $\mathbf{R} \in SO(3)$ and translation vector $\mathbf{t}$ are as follows.

$$\begin{aligned} \mathbf{x} &= [t_x, t_y, t_z, \mathbf{R}]^{\intercal} \\ \mathbf{R} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in SO(3) \\ \mathbf{t} &= \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \end{aligned} \tag{90}$$

The error function $\mathbf{e}_i$ for the i-th point can be written as follows.

$$\mathbf{e}_i(\mathbf{x}) = (\hat{\mathbf{p}}_{t+1,i} - \mathbf{R}\hat{\mathbf{p}}_{t,i} - \mathbf{t}) \tag{91}$$

$$\boxed{\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} \|\mathbf{e}_i(\mathbf{x})\|^2_{\mathbf{M}^{-1}} \\ &= \arg\min_{\mathbf{x}} \|(\hat{\mathbf{p}}_{t+1,i} - \mathbf{R}\hat{\mathbf{p}}_{t,i} - \mathbf{t})\|^2_{\mathbf{M}^{-1}} \end{aligned}} \tag{92}$$

- $d_i = \hat{\mathbf{p}}_{t+1,i} - \mathbf{R}\hat{\mathbf{p}}_{t,i} - \mathbf{t}$: d_i is written out explicitly in the above expression.
- $\|\mathbf{a}\|^2_{\mathbf{B}} = \mathbf{a}^{\intercal}\mathbf{B}^{-1}\mathbf{a}$: Mahalanobis norm
- $\mathbf{M} = (\mathbf{C}_{t+1,i} + \mathbf{R}\mathbf{C}_{t,i}\mathbf{R}^{\intercal})^{-1}$: inverse of the covariance
- $\mathbf{M}^{-1}$: covariance matrix

Unlike conventional ICP variants, GICP is probabilistic, so we optimize the Mahalanobis norm $\|\cdot\|^2_{\mathbf{M}^{-1}}$. In deriving (81), $\mathbf{T} \in SE(3)$ was used, but in implementation one uses $\mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3$. Therefore, covariance propagation for the rotation $\mathbf{R}$ is applied[9].

$$\mathbf{C}_{t+1,i} + \mathbf{T}\mathbf{C}_{t,i}\mathbf{T}^\intercal \quad \rightarrow \mathbf{C}_{t+1,i} + \mathbf{R}\mathbf{C}_{t,i}\mathbf{R}^\intercal \tag{93}$$

The Gauss–Newton derivation is **exactly the same as least-squares point-to-point ICP.**

$$\mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) \approx \mathbf{e}_i(\mathbf{x}) + \mathbf{J}_i\Delta\mathbf{x} \tag{94}$$

The Jacobian for the i-th point is also **exactly the same as least-squares point-to-point ICP.**

$$\boxed{\begin{aligned}\mathbf{J}_i &= \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial[\mathbf{t}, \mathbf{R}]} \\ &= \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial[\mathbf{t}, \Delta\mathbf{w}]} \quad \leftarrow \text{so(3)-based optimization} \\ &= \frac{\partial}{\partial[\mathbf{t}, \Delta\mathbf{w}]}\Big(\mathbf{R}\mathbf{p}_{t,i} + \mathbf{t} - \mathbf{p}_{t+1,i}\Big) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & z'_{t,i} & -y'_{t,i} \\ 0 & 1 & 0 & -z'_{t,i} & 0 & x'_{t,i} \\ 0 & 0 & 1 & y'_{t,i} & -x'_{t,i} & 0 \end{bmatrix} \in \mathbb{R}^{3\times 6}\end{aligned}} \tag{95}$$

- For Lie-theory-based optimization, see the post on error and Jacobian.

Combining the error functions over all points yields the point-cloud error function as follows.

$$\begin{aligned}\mathbf{E}(\mathbf{x}) &= \sum_i^n \|\mathbf{e}_i(\mathbf{x})\|^2_{\mathbf{M}^{-1}} \\ \mathbf{x} &= \arg\min_{\mathbf{x}} \mathbf{E}(\mathbf{x})\end{aligned} \tag{96}$$

Expanding the above in detail yields the formulation (81).

$$\boxed{\begin{aligned}\mathbf{x} &= \arg\min_{\mathbf{x}} \mathbf{E}(\mathbf{x}) \\ &= \arg\min_{\mathbf{x}} \sum_i d_i^\intercal \mathbf{M} d_i\end{aligned}} \tag{97}$$

The point-cloud Jacobian $\mathbf{J}$ and the corresponding $\mathbf{H}, \mathbf{b}$ are aggregated as follows. **Note that in GICP, due to probabilistic modeling, the inverse covariance M is multiplied.**

$$\begin{aligned}\mathbf{H}_i &= \mathbf{J}_i^\intercal \mathbf{M} \mathbf{J}_i \\ \mathbf{b}_i &= \mathbf{J}_i^\intercal \mathbf{M} \mathbf{e}_i\end{aligned} \tag{98}$$

$$\begin{aligned}\mathbf{J} &= \sum_i^n \mathbf{J}_i \\ \mathbf{H} &= \sum_i^n \mathbf{H}_i \\ \mathbf{b} &= \sum_i^n \mathbf{b}_i\end{aligned} \tag{99}$$

The GN solution can be obtained as follows. For readers interested in the derivation, see the post on error and Jacobian.

$$\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b} \tag{100}$$

After computing the optimal increment $\Delta\mathbf{x}^*$ as above, update the original state variable $\mathbf{x}$. Through this update, the source point cloud is gradually registered to the target point cloud.

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^* \tag{101}$$

Repeat the above process until the source point cloud is no longer updated. This procedure is called the Least-squares GICP (3D ver.) algorithm.

Gauss-Newton method (GICP 3D)

1. Set correspondences by assigning to each source point the closest target point using a nearest-neighbor method (e.g., KD-tree).
2. Initialize the covariances $\mathbf{C}_{t,i}, \mathbf{C}_{t+1,i}$. (In fast gicp[9], plane-to-plane covariances are set as defaults)
3. Define the error function as in (81). $\mathbf{e}(\mathbf{x})$
4. Linearize via Taylor expansion and compute the Jacobian. $\mathbf{H} = \mathbf{J}^\intercal\mathbf{M}\mathbf{J}, \mathbf{b} = \mathbf{J}^\intercal\mathbf{M}\mathbf{e}$
5. Set the first derivative to zero. $\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$
6. Compute the increment and apply it to the state. $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^*$
7. Repeat until convergence.

7 References

[1] (youtube) ICP & Point Cloud Registration - Part 2: Unknown Data Association (Cyrill Stachniss, 2021)

[2] (blog) ICP (Iterative Closest Point) and Point Cloud Registration - Jinsol Kim's blog

[3] (blog) Summary of SLAM Lecture 3-2 (ICP algorithm & Unknown Data Association) - taeyoung96's blog

[4] (youtube) [AIX7063] Inclass 19 | Iterative Closest Point lecture

[5] (paper) Chen, Yang, and Gérard Medioni. "Object modelling by registration of multiple range images." Image and vision computing 10.3 (1992): 145-155.

[6] (paper) Rusinkiewicz, Szymon, and Marc Levoy. "Efficient variants of the ICP algorithm." Proceedings third international conference on 3-D digital imaging and modeling. IEEE, 2001.

[7] (paper) Low, Kok-Lim. "Linear least-squares optimization for point-to-plane icp surface registration." Chapel Hill, University of North Carolina 4.10 (2004): 1-3.

[8] (paper) Segal, Aleksandr, Dirk Haehnel, and Sebastian Thrun. "Generalized-icp." Robotics: science and systems. Vol. 2. No. 4. 2009.

[9] (code) SMRT-AIST/fast gicp

[10] (code) ICP Jupyter notebook

8 Revision log

- 1st: 2024-07-06